



UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
-----------------	-------------	----------------------	---------------------	------------------

10/007,581

12/05/2001

Roy F. Brabson

RSW920010223US1

3407

46589 7590 11/05/2007
MYERS BIGEL SIBLEY SAJOVEC P.A.
PO BOX 37428
RALEIGH, NC 27627

EXAMINER

PAN, JOSEPH T

ART UNIT

PAPER NUMBER

2135

MAIL DATE

DELIVERY MODE

11/05/2007

PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Application/Control Number: 10/007,581

Page 2

Art Unit: 2135



UNITED STATES PATENT AND TRADEMARK OFFICE

Commissioner for Patents
United States Patent and Trademark Office
P.O. Box 1450
Alexandria, VA 22313-1450
www.uspto.gov

MAILED

NOV 05 2007

Technology Center 2100

**BEFORE THE BOARD OF PATENT APPEALS
AND INTERFERENCES**

Application Number: 10/007,581

Filing Date: Dec. 5, 2001

Appellant(s): Brabson et al.

David C. Hall

For Appellant (Reg. No: 38,904)

EXAMINER'S ANSWER

This is in response to the Appeal brief filed on July 25, 2007 appealing from the Office action mailed on February 28, 2007. The current application has a co-pending application serial no. 10/007,582, for which the Examiner sent an Examiner's Answer to the Board of Patent Appeals and Interferences on October 10, 2007.

(1) Real Party in Interest

A statement identifying by name the real party in interest is contained in the brief.

(2) Related Appeals and Interferences

The examiner is not aware of any related appeals, interferences, or judicial proceedings which will directly affect or be directly affected by or have a bearing on the Board's decision in the pending appeal.

(3) Status of Claims

The statement of the status of claims contained in the brief is correct.

(4) Status of Amendments After Final

The appellant's statement of the status of amendments after final rejection contained in the brief is correct.

(5) Summary of Claimed Subject Matter

The summary of claimed subject matter contained in the brief is correct.

(6) Grounds of Rejection to be Reviewed on Appeal

The appellant's statement of the grounds of rejection to be reviewed on appeal is correct.

(7) Claims Appendix

The copy of the appealed claims contained in the Appendix to the brief is correct.

(8) Evidence Relied Upon

US 6,141,705 Anand et al. Jun. 12, 1998

US 2003/0014623 A1 Freed et al. Jul. 6, 2001

"The Design of the UNIX Operating System", by Maurice U. Bach,
Prentice Hall Software Series, 1990

(9) Grounds of Rejection

The following ground(s) of rejection are applicable to the appealed claims:

Claims 1-12, 14, 16-18, 20, 22-39 are rejected under 35 U.S.C. 103(a) as being unpatentable over Anand et al. (U.S. Patent No. 6,141,705), hereinafter "Anand", in view of Freed et al. (US Pub. No. 2003/0014623 A1), hereinafter "Freed".

Referring to claim 1:

i. Anand teaches:

A method of performing security processing in a computing network comprising a local unit having an operating system kernel executing at least one application program, comprising:

receiving a first request at the operating system kernel from the application program to initiate a communication with a remote unit (see figure 3, element 140 'application data'; and column 10, lines 27-47 of Anand);

providing a second request from the operating system kernel to a security offload component which performs security handshake processing, the second request directing the security offload component to secure the communication with the remote unit (see e.g. figure 3, element 128 'transport protocol driver, e.g. TCP/IP'; and column 10, lines 27-47 of Anand); and

providing a control function in the operating system kernel for initiating operation of the security handshake processing by the security offload component (see figure 3, element 100 'NIC hardware, e.g. ethernet'; and column 10, lines 27-47 of Anand).

Anand further discloses that "rather than perform certain of the CPU intensive operations on the data packet as it passes through the respective network layers--e.g. checksum calculation/verification, encryption/decryption, message digest calculation and TCP segmentation--those tasks can instead be offloaded and performed at the NIC hardware." (see column 3, lines 39-44 of Anand)

However, Anand does not specifically mention the security handshake processing among the tasks performed by the offload component.

ii. Freed discloses a method for secure communications between a client and a server. The method includes the steps of managing a communication negotiation between the client and the server wherein Freed discloses "Besides authenticating the server to the client, the SSL Handshake Protocol: allows the client and server to negotiate the cipher suite to be used; allows the client and the server to generate symmetric session keys; and establishes the encrypted SSL connection."

Once the key exchange is complete, the client and the server use this session key to encrypt all communication between them.” (see page 1, paragraph [0008], lines 1-7 of Anand, emphasis added)

iii. It would have been obvious to a person of ordinary skill in the art at the time the invention was made to combine the teaching of Freed into the system of Anand to offload the security handshake processing to the offload component.

iv. The ordinary skilled person would have been motivated to have applied the teaching of Freed into the system of Anand to offload the security handshake processing to the offload component, because “As such, there is an advantage in offloading such CPU intensive task to a peripheral hardware device. This would reduce processor utilization and memory bandwidth usage in the host computer, and thereby increase the efficiency, speed and throughput of the overall system.” (see column 2, lines 48-52 of Anand)

Referring to claim 2:

Anand and Freed teach the claimed subject matter: providing a security offload component which performs security handshake, and a control (see claim 1 above). They further disclose executing the provided control function, thereby initiating operation of the security handshake processing (see column 10, lines 27-47 of Anand).

Referring to claim 3:

Anand and Freed teach the claimed subject matter: providing a security offload component which performs security handshake, and a control (see claim 1

above). They further disclose that the operating system Kernel maintains control over operation of the security handshake processing (see column 10, lines 27-47 of Anand).

Referring to claims 4, 7:

Anand and Freed teach the claimed subject matter: providing a security offload component which performs security handshake, and a control (see claim 1 above). They further disclose that kernel does not participate in operation of the security handshake processing (see page 3, paragraph [0034], lines 14-18 of Freed).

Referring to claim 5:

Anand and Freed teach the claimed subject matter: providing a security offload component which performs security handshake, and a control (see claim 1 above). They further disclose specifying information to be used by the security offload component (see figure 4, element 150 'packet extension'; and column 11, lines 8-27 of Anand).

Referring to claims 6, 8:

Anand and Freed teach the claimed subject matter: providing a security offload component which performs security handshake, and a control (see claim 1 above). They further disclose the specified information comprises one or more of: a connection identifier; a security role; cipher suites options, etc. (see page 1, paragraphs [0008], [0010] of Freed).

Referring to claims 9, 30:

Anand and Freed teach the claimed subject matter: providing a security offload component which performs security handshake, and a control (see claim 1

above). They further disclose the completion response from offload component (see page 5, paragraph [0066] of Freed).

Referring to claims 10, 31-32:

Anand and Freed teach the claimed subject matter: providing a security offload component which performs security handshake, and a control (see claim 1 above). They further disclose the conveyed information comprises one or more of: a session identifier, one or more session keys, a sequence number, a cipher suite, etc. (see page 1, paragraphs [0008], [0010] of Freed).

Referring to claim 11:

Anand and Freed teach the claimed subject matter: providing a security offload component which performs security handshake, and a control (see claim 1 above). They further disclose that the operating system kernel maintains control over operation of the security handshake processing, and wherein the operating system kernel provides one or more message segments (see e.g. figure 7, element 237 'Neg. With SSL AD' of Freed).

Referring to claims 12, 14:

Anand and Freed teach the claimed subject matter: providing a security offload component which performs security handshake, and a control (see claim 1 above). They further disclose the random number generation when creating initial handshake message (see page 4, paragraph [0052] of Freed).

Referring to claims 16-17:

Anand and Freed teach the claimed subject matter: providing a security offload component which performs security handshake, and a control (see claim 1 above). They further disclose the certificate and decoding (see page 1, paragraph [0009] of Freed).

Referring to claim 18:

Anand and Freed teach the claimed subject matter: providing a security offload component which performs security handshake, and a control (see claim 1 above). They further disclose the encryption (see page 1, paragraph [0009] of Freed).

Referring to claim 20:

Anand and Freed teach the claimed subject matter: providing a security offload component which performs security handshake, and a control (see claim 1 above). They further disclose the master secret (see page 1, paragraph [0009] of Freed).

Referring to claims 22-23:

Anand and Freed teach the claimed subject matter: providing a security offload component which performs security handshake, and a control (see claim 1 above). They further disclose the master security secrets and the session cryptography keys (see page 1, paragraphs [0008] – [0009] of Freed).

Referring to claim 24:

Anand and Freed teach the claimed subject matter: providing a security offload component which performs security handshake, and a control (see claim 1

above). They further disclose the digitally signing (see page 5, paragraph [0054] of Freed).

Referring to claim 25:

Anand and Freed teach the claimed subject matter: providing a security offload component which performs security handshake, and a control (see claim 1 above). They further disclose validating a digital certificate (see page 1, paragraph [0009], lines 1-8 of Freed).

Referring to claims 26-29:

Anand and Freed teach the claimed subject matter: providing a security offload component which performs security handshake, and a control (see claim 1 above). They further disclose the message authentication code ("MAC") (see page 1, paragraph [0009], last 8 lines of Freed).

Referring to claim 36:

Anand and Freed teach the claimed subject matter: providing a security offload component which performs security handshake, and a control (see claim 1 above). They further disclose preparing the data packet, reserving space in the data packet, and passing the data packet to the offload component (see figure 4, element 142 'network packet'; and column 3, lines 39-44 of Anand).

Referring to claims 37-38:

Anand and Freed teach the claimed subject matter: providing a security offload component which performs security handshake, and a control (see claim 1

above). They further disclose passing control information from the operating system kernel to the security offload component (see column 4, lines 9-12 of Anand).

Referring to claim 39:

Anand and Freed teach the claimed subject matter: providing a security offload component which performs security handshake, and a control (see claim 1 above). They further disclose encrypting the data in the data packet (see column 9, lines 49-50 of Anand).

Referring to claims 33-35:

i. Anand teaches:

A method of performing security processing in a computing network including a local unit having an operating system kernel executing at least one application program, comprising:

providing a security offload component which performs security session establishment and control processing (see figure 3, element 100 'nic hardware'; column 3, lines 31-44 of Anand);

providing a control function in the operating system kernel for initiating operation of the security session establishment and control processing by the security offload component (see column 3, lines 9-23, lines 61-65; and column 4, lines 9-12 of Anand);

receiving a request at the operating system kernel from the application program to initiate a communication with a remote unit (see figure 3, element 140 'application data' of Anand); and

directing the security offload component to secure the communication with the remote unit in response to the request (see column 10, lines 27-47 of Anand).

Anand discloses that "rather than perform certain of the CPU intensive operations on the data packet as it passes through the respective network layers--e.g. checksum calculation/verification, encryption/decryption, message digest calculation and TCP segmentation--those tasks can instead be offloaded and performed at the NIC hardware." (see column 3, lines 39-44 of Anand)

However, Anand does not specifically mention the security session establishment among the tasks performed by the offload component.

ii. Freed discloses a method for secure communications between a client and a server. The method includes the steps of managing a communication negotiation between the client and the server wherein Freed discloses "Besides authenticating the server to the client, the SSL Handshake Protocol: allows the client and server to negotiate the cipher suite to be used; allows the client and the server to generate symmetric session keys; and establishes the encrypted SSL connection. Once the key exchange is complete, the client and the server use this session key to encrypt all communication between them." (see page 1, paragraph [0008], lines 1-7 of Anand, emphasis added)

iii. It would have been obvious to a person of ordinary skill in the art at the time the invention was made to combine the teaching of Freed into the system of Anand to offload the security session establishment to the offload component.

iv. The ordinary skilled person would have been motivated to have applied the teaching of Freed into the system of Anand to offload the security session establishment to the offload component, because "As such, there is an advantage in offloading such CPU intensive task to a peripheral hardware device. This would reduce processor utilization and memory bandwidth usage in the host computer, and thereby increase the efficiency, speed and throughput of the overall system." (see column 2, lines 48-52 of Anand)

(10) Response to Argument

i. Appellant discussed that "Anand is directed to a system by which security processing is performed by an offload component (e.g. a NIC) under the direction and control of an application program, not an operating system kernel. As explained in Anand, "[a]n application executing on the computer system first queries the processing, or task offload capabilities of the NIC, and then selectively enables those capabilities that may be subsequently needed by the application." Anand, Abstract. Furthermore, Anand states that "[o]nce an application has discerned the capabilities of a particular NIC, it will selectively utilize any of the enabled task offload capabilities of the NIC by appending packet extension data to the network data packet that is forwarded to the NIC." Anand, Abstract (emphasis added). Thus, Anand explicitly requires the application program to query the capabilities of an offload processor and then selectively enable those capabilities." (see page 6, last paragraph).

Anand discloses "In Windows NT, a transport protocol driver is a software component that implements a transport driver interface (TDI), or possibly another application-specific interface at its upper edge, to provide services to users of the network." (see column 9, lines 11-15 of Anand, emphasis added). Therefore, Anand discloses that the transport protocol driver is in the operating system kernel.

Anand further discloses "For example, in FIG. 3, the data packet 142 is passed to a software component 144 [i.e., IP Security Driver], which could be implemented separately or implemented as a part of the transport protocol driver itself, that appends a packet extension to the packet 142. Data will be included within in packet extension depending on the particular task that is to be offloaded. For instance, if an IP security function is to be implemented, data that indicates that the NIC 100 should encrypt the data packet in accordance with a specified encryption key would be included. Of course, the software component 144 could append predefined data such that any one of a number of functions, such as those discussed above, would be performed at the hardware level instead of by software components that reside in the network layers. The device driver 116 will extract the information from the packet extension, and then invoke the specified task(s) at the NIC 100." (see figure 3, element 144; and column 10, lines 48-63 Anand, emphasis added).

Anand further discloses that "In a preferred embodiment of the invention, a software implemented method and protocol is provided that allows, for instance, the operating system (OS) to "query" the device drivers (often referred to as "MAC" drivers) of any hardware peripherals (such as a NIC) that are connected to the

computer system. The various device drivers each respond by identifying their respective hardware peripheral's processing capabilities, referred to herein as "task offload capabilities." In the preferred embodiment, once the task offload capabilities of each particular peripheral have been identified, the OS can then enable selected peripherals to perform certain tasks that could potentially be used by the OS. The OS can thereafter request that a peripheral perform the previously enable task, or tasks, in a dynamic, as-needed basis, depending on the then current processing needs of the computer system." (see column 3, lines 9-23 of Anand, emphasis added).

Thus, Anand discloses that the security processing [e.g., encrypting] performed by the offload component is under the control of the operating system kernel.

ii. Appellant stated that "Accordingly, Figure 3 of Anand does not teach or suggest kernel-based offload security processing as recited in Claim 1. In fact, the system of Anand Figure 3 expressly excludes kernel-based offload security processing as recited in Claim 1. For example, the Non-Final Office Action cites element 128 of Anand Figure 3 as teaching the step of providing a second request from the operating system kernel to a security offload component directing the security offload component to secure the communication with the remote unit. However, element 128 of Anand Figure 3 is explicitly labeled a "Transport Protocol Driver," which a skilled person would necessarily understand to be different from an operating system kernel." (see page 7, 1st paragraph, emphasis added).

Anand discloses a functional block diagram illustrating the flow of a data packet through program components (see figure 3, element 128 'transport protocol driver, e.g. TCP/IP', 144 'data packet functions, e.g., IP security driver', 116 'network driver', 100 'NIC hardware e.g., ethernet', of Anand), wherein "in the Windows NT layered networking architecture, a transport protocol driver, or transport, is implemented with an appropriate program method so as to be capable of querying each of the device driver(s) associated with the corresponding NIC(s) connected to the computer." (see column 3, lines 45-50 of Anand, emphasis added).

Anand further discloses "In Windows NT, a transport protocol driver is a software component that implements a transport driver interface (TDI), or possibly another application-specific interface at its upper edge, to provide services to users of the network." (see column 9, lines 11-15 of Anand, emphasis added).

To support the teaching of the ^{driver}~~driver~~ for configuration in an operating system kernel, Maurice Bach is provided. Maurice Bach discloses that drivers, such as device drivers, can be configured in the operating system kernel (see page 20, figure 2.1 'Block diagram of the System Kernel' of "The Design of the UNIX Operating System", by Maurice Bach).

Thus, Anand discloses that the transport protocol driver[figure 3, element 128 of Anand], or transport, is implemented in the Windows NT [i.e., Windows NT Operating System Kernel].

iii. "Appellants submit that the foregoing passage of Anand, which is described by Anand as the "general inventive concept" (Anand, col. 1, line 24), does not mean that in the system of Anand, an operating system kernel can initiate a request to a security offload component to secure a particular communication with a remote unit in response to receiving a request from an application program to initiate a communication with the remote unit, as recited in Claim 1." (see page 7, last paragraph)

Anand discloses the security processing [e.g., encrypting] performed by the offload component is under the control of the operating system kernel (see (i) above). Anand also discloses that it is useful in connection with the offloading of tasks to network interface card (NIC) peripheral device. However, Anand does not discuss in detail the network interface card connection for establishing a communication.

On the other hand, Freed discloses "a method for secure communication between a client and a server. The method includes the steps of managing a communication negotiation between the client and the server," (see abstract, lines 1-4 of Freed, emphasis added).

Freed further discloses the SSL [i.e., Secure Sockets Layer] handshake in "In a manner similar to the embodiment shown in FIG. 5, the client 100 sends a handshaking packet SYN packet to TCP port 443 of SSL accelerator 250 rather than directly to server 300 (at step 202a) The SSL Accelerator device 250 will receive (at step 204a) the SYN packet transmitted by client 100 and may perform functions on packet to enable the SSL acceleration device to continue to perform its SSL proxy functions." (see page 6, paragraph [0068] of Freed).

Therefore, the combination of Anand and Freed disclose an operating system kernel can initiate a request to a security offload component to secure a particular communication with a remote unit in response to receiving a request from an application program to initiate a communication with the remote unit, as recited in Claim 1.

iv. Appellant stated that "Unlike the method recited in Claim 1, the actual control of the particular security processing to be performed is directed by a transport protocol driver, as described, for example, at col 10, lines 27-63 of Anand. Nothing in the cited passage indicates that the operating system kernel can direct a peripheral to secure a particular communication with a particular remote unit" (see page 8, 1st paragraph).

The combination of Anand and Freed disclose an operating system kernel can direct a peripheral to secure a particular communication with a particular remote unit (see (iii) above).

v. Appellant stated that "Appellants submit that interpretation of Anand expressed in the Final Office Action and the Advisory Action clearly ignores the term "environment," which is understood by a skilled person not to refer to the operating system itself, but to the environment created by the operating system in which application programs operate. That is, in the context of computer systems, the term "environment" refers, not to the operating system kernel, but to a computer interface from which various tasks can be performed." (see page 9, last paragraph).

Anand discloses "In a preferred embodiment of the present invention, in the Windows NT layered networking architecture, a transport protocol driver, or transport, is implemented with an appropriate program method so as to be capable of querying each of the device driver(s) associated with the corresponding NIC(s) connected to the computer." (see column 3, lines 45-50 of Anand, emphasis added).

Thus, Anand discloses that the transport protocol driver is implemented in the Windows NT operating system kernel.

vi. Appellant stated that "Moreover, Anand does not suggest the benefits of controlling the operation of a security offload component using a control function in an operating system kernel rather than requiring such control functionality to be implemented in an application program.

Having established that the network driver of Anand is not part of the operating system, Anand teaches that the security functions of a NIC 100 may be invoked by the driver by appending an appropriate packet extension to a data packet." (see page 10, 2nd & 3rd paragraph, emphasis added)

Anand discloses that the security processing [e.g., encrypting] performed by the offload component is under the control of the operating system kernel (see (i) above).

Anand further discloses that the transport protocol driver [figure 3, element 128 of Anand], or transport, is implemented in the Windows NT [i.e., Windows NT Operating System Kernel] (see (ii) above).

(11) Related Proceeding(s) Appendix

No decision rendered by a court or the Board is identified by the examiner in the Related Appeals and Interferences section of this examiner's answer.

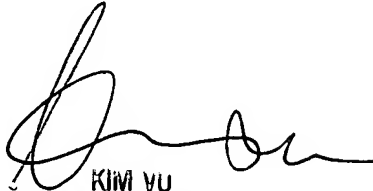
For the above reasons, it is believed that the rejections should be sustained.

Respectfully submitted,

/Joseph Pan/

Conferees:

Ho Song *HS*
Kim Vu *KV*


KIM VU
SUPERVISORY PATENT EXAMINER
TECHNOLOGY CENTER 2100